(12) UK Patent Application (19) GB (11) 2 394 570 (13) A

(43) Date of A Publication 28.04.2004

(21) Application No: 0224445.7

(22) Date of Filing: 21.10.2002

(71) Applicant(s):
NCR International, Inc.
(Incorporated in USA - Delaware)
1700 South Patterson Boulevard, Dayton,
Ohio 45479, United States of America

(72) Inventor(s):
Grant Charles Paton
Gordon David Chisholm
Richard Andrew Han
Alexander Pearson Miller

(74) Agent and/or Address for Service:
F Cleary
NCR Limited, International IP Department,
206 Marylebone Road, LONDON,
NW1 6LY, United Kingdom

(51) INT CL[7]:
G06F 9/445

(52) UK CL (Edition W):
G4A AFLA

(56) Documents Cited:
EP 1096374 A2 WO 2002/046914 A2
US 6117187 A US 20020144248 A1
"APT User's Guide", Jason Gunthorpe, 1998, from:
http://lcmbcu001.epfl.ch/apt-doc/index.html
"Mkpkg: A Software Packaging Tool", Carl Staelin,
August 1998 from:
http://www.hpl.hp.com/techreports/97/HPL-97-125R
1.pdf

(58) Field of Search:
UK CL (Edition V) G4A AFLA AFLB AFLX
INT CL[7] G06F 9/445
Other: ONLINE: EPODOC, PAJ, WPI, various internet
sites

(54) Abstract Title: **Installing software components according to component characteristics**

(57) An aggregate installer 20 for initiating or managing the installation of a plurality of software components 11 in a target device 24. The installer 20 is configured to identify characteristics associated with the components 11 that are to be installed and/or existing software on the target device 24. This can be done using meta-data 12 that is tagged to the components and a component inventory 22 that is provided at the target device 24. Using this data, the aggregate installer 20 devises an installation strategy for the software components that minimises the installation time.

Particularly, the installation process preferably satisfies dependencies either by ordering the installation of components appropriately or by installation of further software components. A further preferable feature is that the installation is performed so as to minimise the number of reboots required.

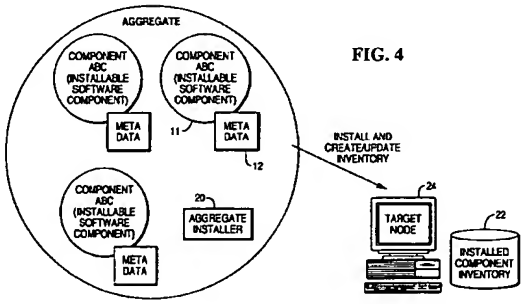

FIG. 4

GB 2 394 570 A

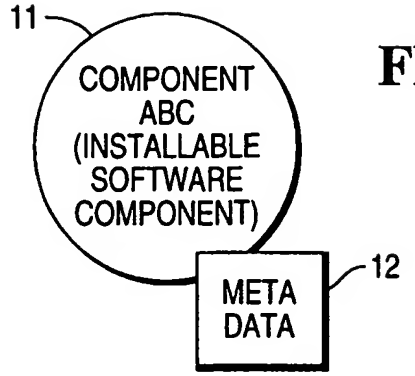
11
COMPONENT ABC (INSTALLABLE SOFTWARE COMPONENT)
META DATA
12

FIG. 1

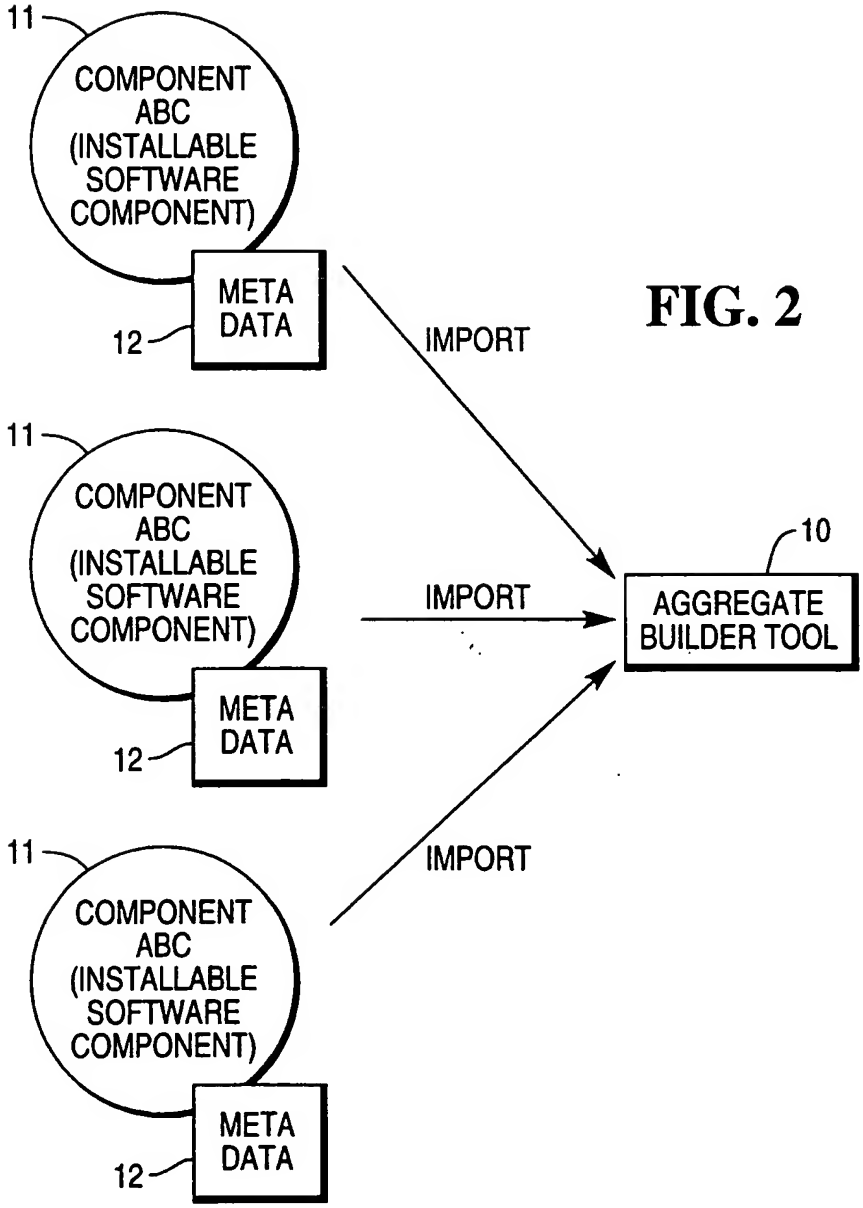
11
COMPONENT ABC (INSTALLABLE SOFTWARE COMPONENT)
META DATA
12
IMPORT
11
COMPONENT ABC (INSTALLABLE SOFTWARE COMPONENT)
META DATA
12
IMPORT
10
AGGREGATE BUILDER TOOL
11
COMPONENT ABC (INSTALLABLE SOFTWARE COMPONENT)
META DATA
12
IMPORT

FIG. 2

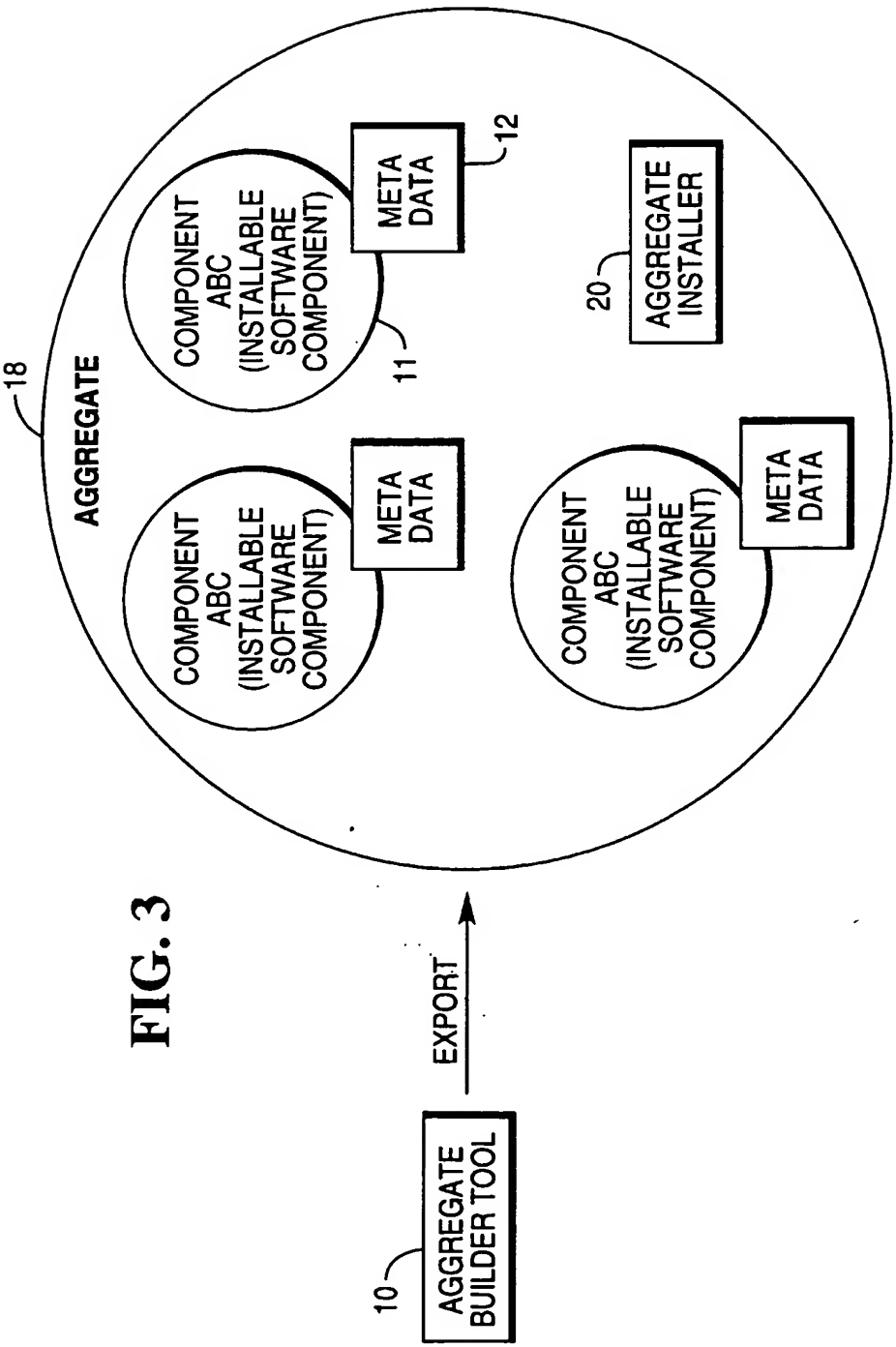
AGGREGATE
18
COMPONENT ABC (INSTALLABLE SOFTWARE COMPONENT)
META DATA
12
11
COMPONENT ABC (INSTALLABLE SOFTWARE COMPONENT)
META DATA
20
AGGREGATE INSTALLER
COMPONENT ABC (INSTALLABLE SOFTWARE COMPONENT)
META DATA
EXPORT
10
AGGREGATE BUILDER TOOL

FIG. 3

FIG. 4

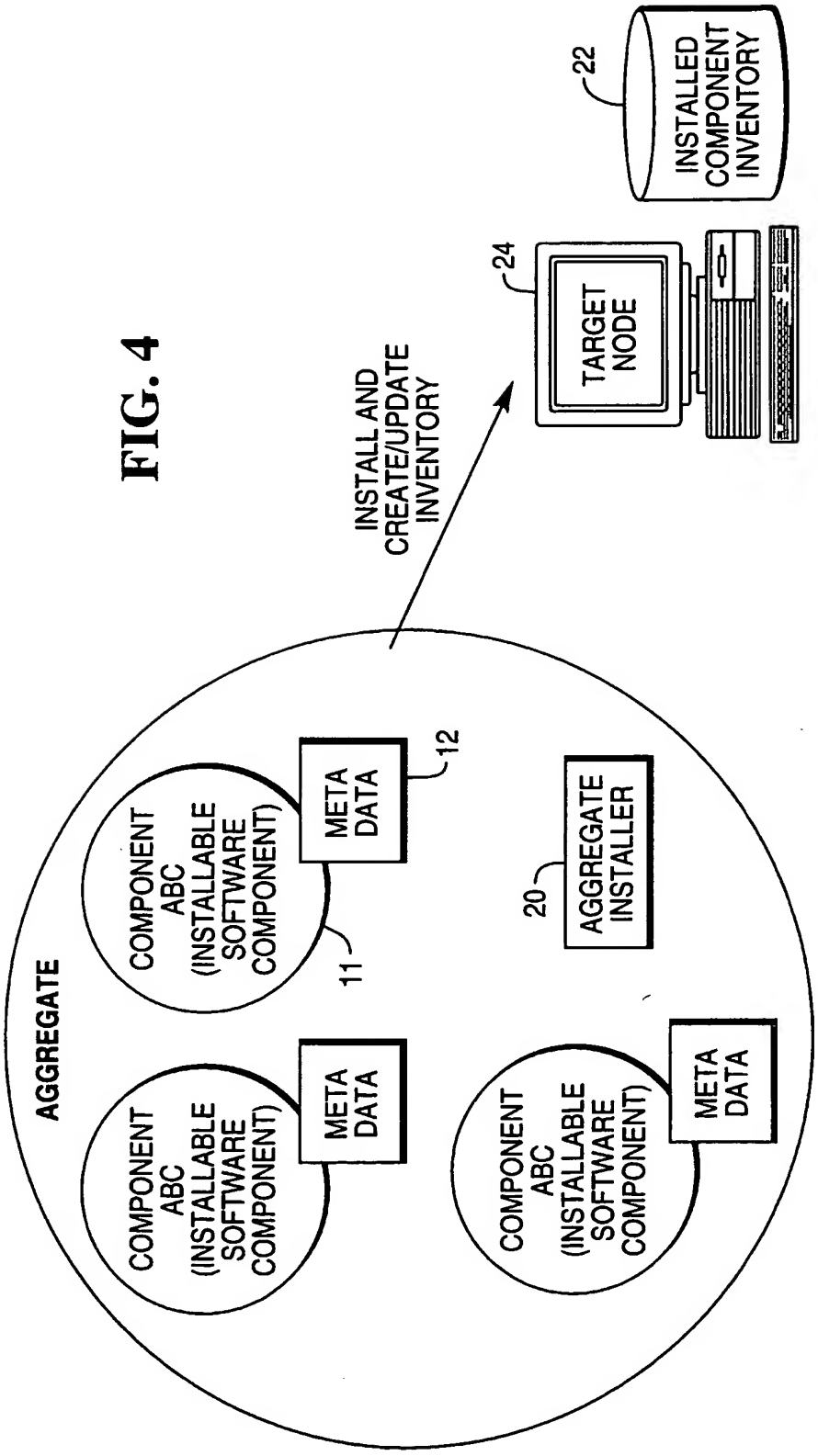
AGGREGATE
COMPONENT ABC (INSTALLABLE SOFTWARE COMPONENT)
META DATA
12
11
COMPONENT ABC (INSTALLABLE SOFTWARE COMPONENT)
META DATA
20
AGGREGATE INSTALLER
COMPONENT ABC (INSTALLABLE SOFTWARE COMPONENT)
META DATA
INSTALL AND CREATE/UPDATE INVENTORY
24
TARGET NODE
22
INSTALLED COMPONENT INVENTORY

FIG. 5

Section	Description
Component Information	Including the component name and version
InstallTimeDependencies	Any other software components (and versions) that must be installed before this component is installed.
Reboot Requirements	When the installation program has completed, what action should be taken in terms of reboots. Possible values are:

Value	Description
No	No reboot required
Yes	A reboot is required at some future time to complete the installation
Automatic	A reboot may or may not be performed by the component installer
ReturnCode	The component installer will return numeric code indicating if it requires a reboot to complete the installation
Immediate	A reboot is required immediately to complete the installation

FIG. 6

Line Number	Present In Inventory	Reboot Requirement	Dependencies Satisfied	Install Order	Install Action
1	Y	any value	any value		Do not install
2	N	any value	No		Install after dependencies installed
3	N	No	Yes	1st	Install
4	N	Yes	Yes	2nd	Reboot to finalise the installation
5	N	ReturnCode	Yes	3rd	Don't reboot unless ReturnCode signifies a reboot is required to finalise the installation
6	N	Automatic	Yes	4th	Component Installer may reboot machine
7	N	Immediate	Yes	5th	Reboot immediately before anything else is installed

FIG. 7

Component	Dependencies	Present in Inventory	Reboot Requirement
A	B	No	Yes
B	H	No	Yes
C	None	No	No
D	None	Yes	Immediate
E	None	No	Yes
F	D	No	ReturnCode
G	None	No	Automatic
H	None	No	No

Software Installation

The present invention relates to a method for installing software components and in particular to a method for optimising such installation.

An installable software component is a piece of software that can be installed on a target node e.g. MS Office, MS Exchange, MS Internet Explorer, Windows Service Pack, Security Hotfix. Conventional software installation technologies have concentrated on managing the installation of such components as single packages of software. More recent technologies have helped create packages containing multiple components. However, these packages tend to have rigid, pre-configured installation scripts that define the order and management of components within the packages.

Problems with installation and configuration are of general concern in all software environments, but this is particularly so in distributed systems in which software may have to be installed on a large number of physically separate nodes or terminals. There are many issues. However, the most significant is that of how to distribute and install the software components onto a target node in the shortest time possible. Installation time is a key issue for business critical systems, for example back office servers, or nodes that provide a valuable customer service,

such as self-service terminals, for example automated teller machines.

An object of the invention is to provide an improved method for optimising the installation of software.

According to one aspect of the present invention, there is provided a method of installing a package of software components in a target device, the method involving identifying characteristics associated with the components that are to be installed and/or existing software on the target device; and devising an installation process that is dependent on the component characteristics and/or existing software identified.

By taking into account the characteristics of the software components that are to be installed and/or software that is already provided on a device, the installation process can be optimised. For example, if it is determined that a component already exists on a device, then there is no need to re-install it, and so it is omitted from the installation process. Also, if it is determined from the component characteristics that say three components require that the device or machine on which they are resident has to be rebooted at some time before they can be used, i.e. before they are fully installed, but not necessarily

immediately, then these can be grouped together and the device or machine rebooted once, rather than three times.

In practice, it has been found for complex systems that by optimising the installation order, the time taken to install new software can be reduced significantly.

Preferably, the step of identifying is carried out at the time of installation.

Preferably, the step of devising is carried out at the time of installation.

The step of identifying may involve comparing each of the software components that are to be installed with existing software components resident at the target device. In the event that there is a match between a new and an existing component, the method further involves preventing the new component from being installed. In the event that there is no match between a new and an existing component, the method may further involve installing the new component. The step of comparing may involve comparing an identifier, for example the name, associated with a component that is to be installed with the corresponding identifiers, for example the names, associated with the existing components.

Preferably, characteristics or identifiers associated with the existing components are listed in an inventory. Preferably, the method further involves maintaining the

inventory of components installed on a target device. The method may further involve up-dating the inventory for the device after the new components are installed.

The characteristics may include a dependency on another component. This means that the other component has to be installed first. In this case, the step of devising an installation process may involve arranging the order of installation of the components to ensure that any dependencies are satisfied.

The characteristics may include reboot information that is indicative of whether the target device has to be rebooted in order to complete installation of a component. In this case, the step of devising an installation process involves arranging the order of installation of the components to ensure that any reboot requirements are satisfied. Preferably, this is done in such a manner to minimise the number of reboots needed.

According to another aspect of the invention, there is provided an installer for installing a plurality of software components in a target device, the installer being configured to identify characteristics associated with the components that are to be installed and/or existing software on the target device; and devise an installation process for

the software components that is dependent on the data and/or existing software identified.

Preferably, the installer is configured to identify the characteristics and/or existing software at the time of installation.

Preferably, the installer is configured to devise the installation process at the time of installation.

The installer may be configured to identify existing software by comparing each of the software components that are to be installed with existing software components resident at the device. In the event that there is a match between a new and an existing component, the installer is configured to prevent the new component from being installed. In the event that there is no match between a new and an existing component, the installer is configured to install the new component. To identify matches between software that is to be installed and existing software, the installer may be configured to compare an identifier, for example the name, associated with a component that is to be installed with corresponding identifiers, for example the names, associated with the existing components.

The existing components, and characteristics thereof may be listed in an inventory. Preferably, the installer is configured to maintain the inventory of components installed

on a device. The installer may be further configured to update the inventory for the device after the new components are installed.

The characteristics may include a dependency on another component. In this case, the installer may be configured to devise an installation process by arranging the order of installation of the components to ensure that any dependencies are satisfied.

The characteristics may include reboot information that is indicative of whether the target device has to be rebooted in order to complete installation of a component. In this case, the installer is configured to devise an installation process by arranging the order of installation of the components to ensure that any reboot requirements are satisfied. Preferably, this is done in such a manner to minimise the number of reboots needed.

According to yet another aspect of the invention, there is provided a computer program, preferably on a data carrier or a computer readable medium, for installing a plurality of software components on a target device, the computer program having code or instructions for identifying data associated with the components that are to be installed and/or existing software on the target device; and devising an order for

installing the software components that is dependent on the data and/or existing software identified.

Various aspects of the invention will now be described by way of example only, and with reference to the following drawings, of which:

Figure 1 is a simplified diagrammatic representation of a software component and its associated meta-data;

Figure 2 is a schematic diagram of a system for grouping software components into packages and installing these at remote locations;

Figure 3 is a schematic view of a package that is exported from the system of Figure 2;

Figure 4 is a schematic view of the package of Figure 3 being installed on a target device;

Figure 5 is a table setting out types of meta-data that could be associated with software components in the package of Figures 3 and 4 and used in the installation optimisation process;

Figure 6 is a representation of an algorithm that is used to optimise the installation of the packages of software of Figures 3 and 4, and

Figure 7 is a table of characteristics for a package of software components that has to be installed at a client terminal.

The present invention relates to a system and method for optimising the installation of software components. To achieve this, it is necessary to address particular problems that arise when trying to distribute a collection of software components together. For example, which component should be installed first, what are the interdependencies, when does the system have to be rebooted, is a particular component already installed etc?

For the purposes of this application a piece of software that can be installed on its own is referred to as a component. A group of software components is referred to as an aggregate or package. An aggregate may a grouping of components and/or other aggregates. Because aggregates can contain other aggregates, an aggregate represents a tree like structure identifying different groupings of components.

In the present application, each component is associated with metadata that describes various component attributes, as shown in Figure 1. The metadata may describe many aspects of the software component including the following: component name; component description; windows add/remove programs key that can be used to de-install the component; compatibility with previous version of the component, including which previous versions the new version

can directly upgrade from; the type of installation mechanism the component uses; special installation requirements, such as 'must be last in installation sequence'; windows administrative privileges required by component; operating system dependencies, for example NT version, Service Pack requirements; installation time dependencies of the component, which are typically declared in the form of the names and versions of other components; run time dependencies of the component, declared in the form of the names and versions of other components; target environments supported by the component, for example, self service terminal, development PC, simulation environment; command lines to install, upgrade and de-install the component for each specific environment supported; machine reboot requirements of each component installation, for example whether no reboot is required or a reboot is required sometime in the future to complete installation or a reboot is required immediately to complete installation or a reboot may or may not be needed to complete installation; and disk space requirements of each component installation, among others. Each of the software components to be installed on the target requires this metadata.

In order to form an aggregate of desired software components, once the appropriate metadata is created and

tagged to the components of interest, these components are selected by the system developer and imported into an aggregate builder tool 10, as shown in Figure 2. This builder tool 10 enables components 11, together with their meta-data 12, to be grouped together into aggregates and allows aggregate installation parameters to be pre-set for those aggregates or components. To do this, the aggregate builder tool 10 imports the selected components 11 and aggregates them into an area called the aggregate customisation archive. Components 11 can be imported into the aggregate customisation archive from any disk or storage location accessible to the aggregate builder tool 10. The aggregate builder tool 10 is adapted to allow aggregates 18 within the aggregate customisation archive to be edited so that they can be customised for particular applications. Because of this, the aggregate builder tool 10 can help create and manage multiple software packages and also allow the content of those packages to be edited, quickly and easily. Using the aggregate builder tool, components 11 can be reused easily and effectively with little or no additional effort on the part of the component developer.

Once an aggregate 18 is created, the aggregate builder tool 10 includes an extra piece of software 20 in the aggregate 18. This software will be referred to as the

aggregate installer 20, see Figure 3. Once this is done, the aggregate 18, together with its installer 20 is exported to the target terminal 24, as shown in Figure 4. This can be done manually or by downloading the software over a communications network.

The function of the aggregate installer 20 is to install the software components in the aggregate 18; create and maintain a software component inventory 22 on the target node 24 and optimise the installation of the aggregate 18 based on the component metadata and the software component inventory 22 located at the target node 24. The software inventory 22 includes the metadata for all of the components installed on the target terminal. In order to ensure that this is kept up to date, the installer 20 is adapted to enter the metadata associated with each of the components it installs once the installation process is completed.

It should be noted that the details of each component's installation are defined by the component's metadata. The aggregate installer 20 itself does not perform any actual component installations, it only initiates and manages them. Because of this any required installation method can be used as long as it conforms to the rules for component definition. For example, APTRA Advance NDC 2.3 uses Installshield to install. As long as the component

installation mechanism can be described using the component metadata, a component definition could be written to encapsulate the installation of the component. This component could then be used by the aggregate builder to create an installable aggregate that contains the APTRA Advance NDC 2.3 component. Other valid installation methods are Windows Installer, Installshield, INF file etc.

The aggregate installer 20 is adapted to optimise the installation routine and devise an installation process or strategy. One of the most important features in doing this is determining the order in which components are to be installed in order to avoid unnecessarily repeating steps, such as rebooting, and installing components that are already available on the terminal. To this end, the aggregate installer 20 includes an optimisation algorithm that devises an installation script for setting the order of installation of the components.

The optimisation algorithm is based on three factors: the state of the inventory of installed components on the target system; the install time dependencies of the component meta-data and the reboot required flag of the component meta-data, which indicates whether installation requires the system to be re-booted. Hence, when the exported software package is to be installed, the installer

20 goes through all of the meta-data 12 for all of the components 11 at install time, as well as the inventory 22 for the target terminal 24 and uses this data to dynamically create an optimised installation script. Typically, this involves identifying firstly whether any of the components that are to be installed actually exist on the terminal already and if they do prevent these from being installed; then identifying any dependencies and setting an initial installation order that guarantees these and finally checking the rebooting status of the components and determining an installation order that ensures that all reboot requirements are met, whilst minimizing the number of such reboots needed. Once the optimal order of installation is determined, an installation script is dynamically created to install the components in that order.

In order for the optimization algorithm to determine an optimal installation process or strategy, it has to have access to certain information relating to the components within the aggregate. As mentioned previously, this information is included in the meta-data 12 associated with each component 11. Figure 5 shows a table of the particular types of information that can be used by the algorithm. This includes component information, for example, the component name and version number; install time

dependencies, which indicate other software components that have to be installed before this component is installed; and reboot requirements, which indicate what action has to be taken in terms of rebooting the system in order to ensure that the component is installed. Various reboot requirement values are provided including "No"; "Yes"; "Automatic"; "ReturnCode" and "Immediate". These will be described in more detail with reference to Figure 6.

Figure 6 shows various combinations of component/system characteristics that the optimization algorithm may come across, as well as the install action that is to be carried out in the event that the algorithm identifies a component having these specific combinations of features. From line one of the table of Figure 6, it can be seen that this component is already installed in the target device, any reboot requirement can be used and there are no dependencies. In this case, the action for the algorithm is "do not install". This is because there is no need to include another version of the component if there is already one in existence that can be shared. Hence, should the algorithm encounter a component having these characteristics, no action is taken.

In line two of the table of Figure 6, the component is not in the inventory, any reboot value can be used and the

dependency requirements have not been satisfied. In this case, the optimization algorithm is adapted to install the component, but only after the components in the installation order list are sorted so that the dependencies are satisfied. Hence, in the event that the optimization algorithm encounters components that are not in the correct order to ensure that their dependencies are set-up, the first step that is taken by the algorithm is to order the components so that the dependencies can be guaranteed.

In line three of the table of Figure 6, the component illustrated is not in the inventory, the reboot requirement is "No", which means that no reboot is needed, and the dependency requirements have been satisfied. In this case, the optimization algorithm installs the component immediately. Because of the lack of dependency on other components and the fact that there is no need for rebooting, this type of component can be installed before others.

In line four of the table of Figure 6, the component is not in the inventory, the reboot requirement is "Yes", which means that a reboot is needed at some stage before installation of the component can be completed, and the dependency requirements have been satisfied. In this case, the optimization algorithm has to ensure that a reboot is effected at some stage in order to install the component.

This type of component is installed by the algorithm after those components that do not require a reboot to complete installation.

In line five of the table of Figure 6, the component is not in the inventory, the reboot requirement is "ReturnCode" and the dependency requirements are satisfied. The "ReturnCode" requirement is an install process exit code. This is generated when the set up for the component terminates. It includes a code or identifier that is indicative of whether a reboot is required. In this case, the optimization algorithm installs the component, but does not reboot unless the ReturnCode signifies that a reboot is needed to finalise the installation. The return code may indicate that a reboot is needed in the event that the component that is being installed has become locked by another process and the machine must be rebooted to unlock the file. The algorithm is adapted to cause this type of component to be installed after those that need a reboot at some stage before the component is properly installed.

In line six of the table of Figure 6, the component is not in the inventory, the reboot requirement is Automatic and the dependency requirements have been satisfied. In this case, for the reboot requirement "Automatic", the component and may or may not reboot the terminal, rather

than this being controlled by the aggregate installer 20. This particular reboot requirement is needed for components that are installed using earlier versions of installers, such as InstallShield, that may or may not reboot the machine and do not support a process exit code to inform the aggregate installer 20 that a reboot is needed. By providing this code, it means that old components can be re-used without alteration. Before allowing the installation of a component of this type, the aggregate installer 20 sets up the system so that if a reboot occurs, the aggregate installer 20 restarts after the reboot and carries on where it left off. The algorithm is adapted to install components having the reboot requirement "Automatic" after those that have the reboot requirement "ReturnCode".

In line seven of the table of Figure 6, the component is not in the inventory, the reboot requirement is Immediate and the dependency requirements have been satisfied. In this case, the optimization algorithm installs the component and immediately reboots the system before anything else is installed. The algorithm is adapted to install components having these characteristics last.

As an example, consider the simple aggregate shown in Figure 7. Based on the RebootRequirement information alone, and using conventional techniques, this installation should

take between four and six reboots to complete. However, applying the optimisation algorithm to identify whether any of the new components are already installed shows that component D is included in the inventory for the target device and so is already installed. This means that D does not have to be installed again. Identifying this reduces by one the number of reboots that have to be done.

Having identified existing components, the algorithm is adapted to investigate dependencies. To do this, it checks the metadata for each of the components to determine whether its installation is dependent on the earlier installation of other components. From Figure 7, it can be seen that for components C, D, E, G and H there are no dependencies. In contrast, component A is dependent on component B. This means that A has to be installed before B. In addition, component B is dependent on component H. This means that H has to be installed before component B. Finally, component F is dependent on D. In this case, however, since D is already installed on he target terminal, it does not need to be re-installed before F.

Once the dependencies are checked and satisfied, the algorithm investigates the reboot flags for each component. This shows that no reboot is needed to install C. Hence, C has all of the characteristics of the component of line

three of Figure 8 and so can be installed first. The same is true for component H, which can be installed directly after C. But A, B and E are only installed when the system is, at some time, rebooted. Since A is dependent on B and the system has to be rebooted to ensure that B is installed, this means that B and E should be installed before A and additionally that the system has to be rebooted to install B before A can be installed. For F, the reboot requirement is "ReturnCode". In the present case, it is assumed that the installer identifies that a reboot is necessary. Because of the status of the reboot requirement for F, this means that it has to be installed after C, H, E and B. For component G, the reboot requirement is Automatic, which means that the algorithm places it behind the other components in the installation order list.

Taking into account its various checks, for the case of the aggregate of Figure 7, the optimsation algorithm gives the following installation sequence:

1. Don't install D;
2. Install C
3. Install H
4. Install E
5. Install B
6. Install F

7. Install G

8. Reboot

9. Install A

10. Reboot

This sequence requires only two reboots to perform the complete installation, rather than the 4 to 6 as would be required using conventional installation techniques. In practice, this represents a significant time saving. Obviously more complex installations give greater potential for savings.

Once the installation order is identified, the algorithm creates an installation script for initiating and controlling installation of each of the components in the specified order. This script causes each of the components to be installed in the correct order and causes reboots to occur, when necessary in the overall installation sequence. Once the components are installed, the installation script causes the software component inventory 24 to be up-dated.

By dynamically creating installation scripts at installation time independently of the package itself, there is provided a simple and effective mechanism for ensuring that the installation process is optimised to suit the target environment as well as be tailor to take into account the multiple components that are to be installed, without

any prior knowledge of the specific installation environment. An advantage of this is that installations times can be significantly reduced. In addition, by moving away from the conventional idea of fixed, inflexible installation scripts, it means that the installation script algorithm itself can be up-dated without altering the package. This means that the skill level and training required by application developers for software packaging and installation can be reduced, thereby saving time and money. Using the aggregate builder, an organisations entire set of software solutions can be delivered and installed using one single installation technology even if the constituent software components use different installation technologies.

A skilled person will appreciate that variations of the disclosed arrangements are possible without departing from the invention. Accordingly, the above description of a specific embodiment is made by way of example only and not for the purposes of limitation. It will be clear to the skilled person that minor modifications may be made without significant changes to the operation described.

Claims

1. A method of installing a plurality of software components in a device, the method involving identifying characteristics associated with the components that are to be installed and/or existing software on the target device; and devising an installation process that is dependent on the data and/or existing software identified.

2. A method as claimed in claim 1, wherein the step of identifying is carried out at the time of installation.

3. A method as claimed in claim 1 or claim 2, wherein the step of devising is carried out at the time of installation.

4. A method as claimed in any of the preceding claims, wherein the step of identifying involves comparing each of the software components that are to be installed with existing software components resident at the device and preventing the new component from being installed if an identical component already exists at the device.

5. A method as claimed in claim 4, wherein the step of comparing involves comparing an identifier, for example the

name, associated with a component that is to be installed with corresponding identifiers, for example the names, associated with the existing components.

6. A method as claimed in claim 5 comprising storing identifiers associated with existing components in an inventory.

7. A method as claimed in claim 6 further involving maintaining the inventory of components installed on a device.

8. A method as claimed in claim 7 further involving updating the inventory for the device after the new components are installed.

9. A method as claimed in any of the preceding claims, wherein the characteristics include a dependency on another component.

10. A method as claimed in claim 9, wherein the step of devising an installation process involves arranging the order of installation of the components to ensure that any dependencies are satisfied.

11. A method as claimed in any of the preceding claims, wherein the characteristics include reboot information that is indicative of whether the target device has to be rebooted in order to complete installation of a component.

12. A method as claimed in claim 11, wherein the step of devising an installation process involves arranging the order of installation of the components to ensure that any reboot requirements are satisfied.

13. A method as claimed in claim 12, wherein the step of arranging is done in such a manner to minimise the number of reboots needed.

14. An aggregate installer for initiating or managing the installation of a plurality of software components in a target device, the installer being configured to identify characteristics associated with the components that are to be installed and/or existing software on the target device; and devise an installation process for the software components that is dependent on the data and/or existing software identified.

15. An installer as claimed in claim 14 configured to identify the characteristics and/or existing software at the time of installation.

16. An installer as claimed in claim 14 or claim 15 configured to devise the installation process at the time of installation.

17. An installer as claimed in any of claims 14 to 16 configured to identify existing software and compare each of the software components that are to be installed with existing software components identified and prevent the new component from being installed if an identical component already exists at the device.

18. An installer as claimed in claim 17 configured to compare an identifier, for example the name, associated with a component that is to be installed with identifiers, for example the names, associated with the existing components.

19. An installer as claimed in claim 17 or claim 18 configured to maintain an inventory of components or characteristics of such components installed on a device.

20. An installer as claimed in claim 19 further configured to up-date the inventory for the device after the new components are installed.

21. An installer as claimed in any of claims 14 to 20, wherein the characteristics include a dependency on another component and the installer is configured to devise an installation process by arranging the order of installation of the components to ensure that any dependencies are satisfied.

22. An installer as claimed in any of claims 14 to 21, wherein the characteristics include reboot information that is indicative of whether the target device has to be rebooted in order to complete installation of a component and the installer is configured to devise an installation process by arranging the order of installation of the components to ensure that any reboot requirements are satisfied.

23. A system for installing a plurality of software components in a computer based system, the system comprising means for identifying characteristics associated with the components that are to be installed and/or existing software

on the target device; and means for devising an installation process that is dependent on the data and/or existing software identified.

24. A system as claimed in claim 23, wherein the means for identifying are operable to identify characteristics and/or existing software at the time of installation.

25. A system as claimed in claim 23 or claim 24, wherein the means for devising are operable to devise the installation process at the time of installation.

26. A computer program, preferably on a data carrier or a computer readable medium, for installing a plurality of software components on a target device, the computer program having code or instructions for identifying data associated with the components that are to be installed and/or existing software on the target device; and devising an order for installing the software components that is dependent on the data and/or existing software identified.

Application No:	GB 0224445.7	**Examiner:**	Paul Jefferies
Claims searched:	1-26	**Date of search:**	11 June 2003

Patents Act 1977 : Search Report under Section 17

Documents considered to be relevant:

Category	Relevant to claims	Identity of document and passage or figure of particular relevance	
X	1-26	EP 1096374 A2	(CITICORP) See abstract and paragraphs [0061] - [0063].
X	1-10, 14-21, 23-26	WO 02/46914 A2	(ADUVA) See abstract, figure 1 and page 6.
X	1-10, 14-21, 23-26	US 2002/0144248 A1	(FORBES et al.) See abstract and figure 3A.
X	1-10, 14-21, 23-26	US 6117187	(STAELIN) See abstract and figure 1.
X	1-10, 14-21, 23-26	"APT User's Guide", Jason Gunthorpe, 1998, from: http://icmbcu001.epfl.ch/apt-doc/index.html	
X	1-10, 14-21, 23-26	"Mkpkg: A Software Packaging Tool", Carl Staelin, August 1998 from: http://www.hpl.hp.com/techreports/97/HPL-97-125R1.pdf	

Categories:

X	Document indicating lack of novelty or inventive step	A	Document indicating technological background and/or state of the art.
Y	Document indicating lack of inventive step if combined with one or more other documents of same category.	P	Document published on or after the declared priority date but before the filing date of this invention.
&	Member of the same patent family	E	Patent document published on or after, but with priority date earlier than, the filing date of this application.

Field of Search:

Search of GB, EP, WO & US patent documents classified in the following areas of the UKC[V]:

G4A

Worldwide search of patent documents classified in the following areas of the IPC[7]:

G06F

The following online and other databases have been used in the preparation of this search report:

Online: WPI, EPODOC, JAPIO, Internet